Lecture 6: Autocovariance and decorrelation time scales

Recap

Lecture 4 looked at variance and covariance, and Lecture 5 explored the statistics of a random walk as well as quantifying sampling errors. We derived the fact that the standard error of the mean si $\sigma/\sqrt{N}$ and that from real observations, standard deviations and variances need to be computed using $N-1$ rather than N—in essence, we lose a degree of freedom when we compute the mean, and we have to take that into account. Now we're going to plunge into the details of lagged autocovariances.

Correlated uncertainties and degrees of freedom

We've been considering idealized situations in which random variables x_n are completely independent, so that the covariance of x is of the form:

$$\langle x'_n x'_m \rangle = \sigma^2 \delta_{nm}. \tag{1}$$

This works well if your random variables come from a random number generator, but not so well if they come from geophysical quantities that are sampled more rapidly in time (or space) than the system actually evolves. In reality variables that are closely spaced in time or space tend to be correlated. If the statistics are stationary, then the covariance between measurements might depend only on their separation. For example:

$$\langle x'_n x'_m \rangle = \sigma^2 \rho(|n-m|) \tag{2}$$

We can compute the autovariance of x with itself at multiple lags as a lagged covariance. (In Matlab you can do this with the function "xcov".) There are some things to keep in mind.

1. For a record of finite length, the number of data pairs that we can average to compute the autocovariance varies with the size of the lag. For a record with N values, at zero lag, the autocovariance is an average of N data pairs. At lag n, it is an average of $N-n$ data pairs. This means that our estimate of the autocovariance has larger uncertainty for larger lags.

2. To deal with the uncertainties in pairs, the default in Matlab is to compute a "biased" autocovariance, by scaling the autocovariance by $(N-|n|)/N$ so that the autocovariance tapers to zero at large lag.

3. You can also compute an "unbiased" autocovariance, which will compute the average covariance based on the available number of pairs. This works well near zero lag, when you can average over many pairs, but it has the downside of producing a highly uncertain covariance for large lag.

4. The autocovariance has units, corresponding to the units of the data squared. For many calculations, we want the autocorrelation rather than the autocovariance—that is we want an autocorrelation that varies between -1 and +1 and that is unitless. In Matlab, we can compute this using the "coeff" flag, which produces values consistent with correlation coefficients.

Now let's think about an example. Consider the case of random variables that are each repeated 10 times in our data record. In class we produced a data record using the following lines of Matlab code:

```
% first define a set of random numbers
N=1000;
x=randn(N,1);
% now make them repeat 10 times
y=x*ones(1,10);

% plot this out:  is it right?
plot(y(:))
%   no, this case visibly repeats the entire record 10 times.
%   that's just because when we convert from a matrix to a vector,
%   Matlab does this by concatenating the columns of the matrix.
%
%  If we were using python, the data storage would be inverted.

% take the transpose and plot a subset
z=y';
plot(1:200,z(1:200))
xlabel('time step','FontSize',14)
ylabel('random data value','FontSize',14)
h=gca;
set(h,'FontSize',14)
shg
```

or

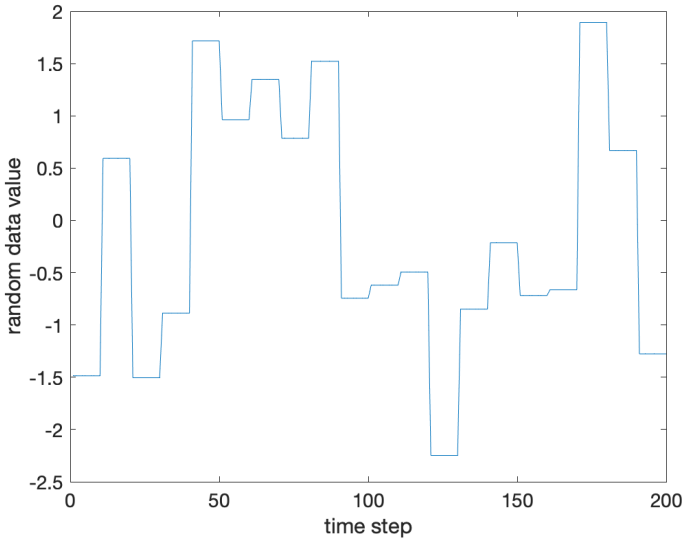
```
import numpy as np
import numpy.matlib
import matplotlib.pyplot as plt

# first define a set of random numbers
N=1000
x=np.random.normal(size=[N,1]);
# now make them repeat 10 times
y=np.matlib.repmat(x,1,10)

# plot this out:  is it right?
plt.plot(y.flatten())
# we can't tell from a single plot but enlarging the axes helps

# here's a zoom
plt.plot(y.flatten())
plt.axis([0,300,-3,3])
plt.xlabel('time step')
plt.ylabel('time step')
# this looks good.
# If we'd had trouble, we could have taken the transpose of y before inverting.
```

Once we have our data, which clearly have correlated consecutive values, we can compute the autocorrelation:




```
% compute the autocorrelation
Nbig=N*10;
plot(-Nbig+1:Nbig-1,xcov(z(:),'coeff'))
xlabel('lag','FontSize',14)
ylabel('autocorrelation','FontSize',14)
h=gca;
set(h,'FontSize',14)
```

or

```
# compute the autocorrelation
Nbig=N*10
xcov=np.correlate(y.flatten(),y.flatten(),mode='full')
plt.plot(range(-Nbig+1,Nbig),xcov/xcov[Nbig-1])
plt.xlabel('lag')
plt.ylabel('autocorrelation')
```

or alternatively, use

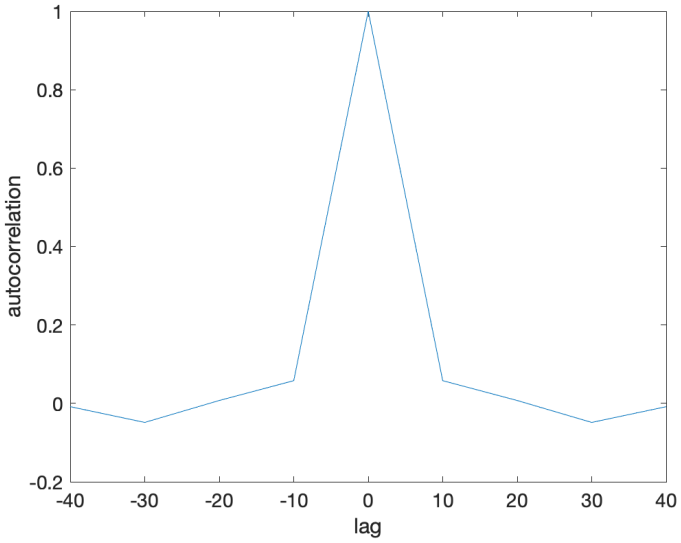
```
import statsmodels.tsa.stattools
xcov=statsmodels.tsa.stattools.acf(y.flatten(),nlags=Nbig)
```

The zero lag version of this is hard to read, but let's zoom in on the center of the distribution:

```
axis([-40 40 -.2 1])
```

or

```
plt.plot(range(-Nbig+1,Nbig),xcov/xcov[Nbig-1])
plt.xlabel('lag')
plt.ylabel('autocorrelation')
plt.axis([-40,40,-.2,1.1])
```



This shows that the autocovariance is effectively a triangle rising from 0 at $\tau = -10$ to 1 at $\tau = 0$ and then falling to 0 at $\tau = 10$. This might not be what you'd expected, but it's not entirely surprising: the autocovariance represents a convolution of the data with itself, and the convolution of a boxcar distribution with itself is a triangle.

How many data steps should it take to obtain an independent value? In this case we know that every 10th value is independent, so our "decorrelation scale" is 10 measuremewnts, and the number of degrees of freedom $Neff$ is the total number of values in z ($Nbig$) divided by 10, which is N.

How can we show this more generally? There are a number of heuristic approaches that people use, but a good formalism is to integrate the autocovariance.

$$\text{"decorrelation" scale} = \tau_{eff} = \int_{-\infty}^{\infty} \rho(t)\, dt. \tag{3}$$

In this idealized example, that would yield:

$$\begin{aligned}
\tau_{eff} &= \int_{-\infty}^{\infty} \rho(t)\, dt && (4)\\
&= \int_{-10}^{0} \left(1 + \frac{t}{10}\right) + \int_{0}^{10} \left(1 - \frac{t}{10}\right) && (5)\\
&= \left(t + \frac{t^2}{20}\right)\Big|_{-10}^{0} + \left(t - \frac{t^2}{20}\right)\Big|_{0}^{10} && (6)\\
&= \left(10 - \frac{100}{20}\right) + \left(10 - \frac{100}{20}\right) = 10. && (7)
\end{aligned}$$

Autocovariance and degrees of freedom

First recall that the autocovariance is the convolution of a data set with itself. One definition of the autocovariance is:

$$C_{xx}(r) = \frac{\sum_{i=1}^{N-r} \sum_{j=1+r}^{N} x'_i x'_j}{N - r}. \tag{8}$$

This provides an "unbiased" estimate of the autocovarince—unbiased because we divide each covariance by the number of data pairs that are available at separation r.

The unbiased estimator is ill-behaved for large lags, because we have few data pairs to average. Thus more typically we use a "biased" estimator:

$$C_{xx}(r) = \frac{\sum_{i=1}^{N-r} \sum_{j=1+r}^{N} x_i' x_j'}{N}. \tag{9}$$

This tapers to zero for large lags.

The autocorrelation resembles the autocovariance, except that it is normalized by the data variance:

$$\rho_{xx}(r) = \frac{\sum_{i=1}^{N-r} \sum_{j=1+r}^{N} x_i' x_j'}{N\sigma^2}, \tag{10}$$

where σ is the standard deviation of x.

If we want to figure out how many degrees of freedom we have in a data set of length N, we need to figure out how redundant adjacent data points are—that is we need a "decorrelation" scale. There are three common ways to estimate this:

1. The first zero crossing—that is the first point when the autocorrelation decreases from a positive value to a negative value. This is conceptually easy, but fraught when the data are noisy. Unfortunately, it's easy to show that data with very different autocorrelation structure, and correspondingly very different decorrelation scales, can nonetheless have the same zero crossing.

2. The full-width at half-maximum of the autocorrelation. This is potentially a slightly less noisy-prone approach to obtaining a quick estimate of a decorrelation scale. For a well-behaved decorrelation, such as a triangle autocovariance, this will produce clean results.

3. The integral of the autocovariance ($\int_{-\infty}^{\infty} \rho(r)\, dr$) is a robust measure of decorrelation, though it also is subject to interpretation depending on noise and sampling limitations. We'll pursue this approach in more detail.

If we measure N data points, then the effective degrees of freedom should be N divided by the decorrelation scale:

$$N_E = \frac{N}{\sum_{n=-N}^{N} \rho(|n|)}. \tag{11}$$

In writing this, I'm assumed that $\rho(n)$ is well behaved at $\pm N$. If I used an unbiased estimator, then I'd want to suppress the uncertain values at high n

$$N_E = \frac{N}{\sum_{n=-N}^{N} \left(1 - \frac{|n|}{N}\right) \rho(|n|)}. \tag{12}$$

But in Matlab, the "xcov" default is the "biased" estimator, and we can ignore this. In python, "np.correlate" simply computes the unnormalized convolution of the data with themselves, so when you normalize by the zero lag value, it is a biased estimator.

So consider the case when data are random white noise, and adjacent points are uncorrelated. Then $\rho(n) = \delta(n)$ which is 1 only when $n = 0$, and

$$N_E = \frac{N}{\sum_{n=-N}^{N} \delta(|n|)} = N. \tag{13}$$

In other words, in this case, all data are independent.

More realistically, geophysical data tend to be correlated over time and space. Usually if we sum from $-N$ to $+N$, our results will be contaminated by noise, so we actually want to sum only over the center portion of the autocovarince.

$$N_E = \frac{N}{\sum_{n=-l}^{l} \rho(|n|)}, \tag{14}$$

where $l < N$. The trick is to decide on l: how many values should we sum? If we lack any other information, we can just test all possible l and look at the range of N_E values that we find. Usually we want to be as conservative as possible, so we assume the smallest possible N_E—that is the largest possible decorrelation scale.

If we write this continuously, we can express the values as integrals instea of sums. In other words the observed mean of x is:

$$\{x\} = \frac{1}{T} \int_0^T x(t)\, dt. \tag{15}$$

And in continuous form,

$$N_E = \frac{T}{\int_{-T}^{T} \rho(t)\, dt}. \tag{16}$$

The expected value of the variance is:

$$\begin{aligned} E_2 &= \frac{1}{T^2} \int_0^T \int_0^T \langle x'(t_1) x'(t_2) \rangle \, dt_1 \, dt_2 && (17) \\ &= \frac{\sigma^2}{T} \int_{-T}^{T} \rho(|t|)\, dt && (18) \\ &= \frac{\sigma^2}{N_E}. && (19) \end{aligned}$$

This means that:

$$T_{eff} = \frac{T}{N_E} = \int_{-T}^{T} \rho(|t|)\, dt. \tag{20}$$

In class we looked at a range of examples to understand how different studies have analyzed decorrelation, and how decorrelation can be sensitive to the large-scale variations in the data record.